Amendment to the Specification

Please title the application to "Instruction/Data Protection Employing Derived Obscuring Instruction/Data".

Page ~~11~~ 10, paragraph starting now line 17

In the embodiment of the invention shown in Figure 2 the transformation are concatenated so that any block C_N is generated by a series of mathematical transformation T such that $C_N = T_N(T_{N-1}(\ldots(T_3(T_2(C_1)))\ldots))$, where C_1 is an initial block of obscuring instructions selected from~~form~~ the obscuring code bank. By concatenating the transformations, it is possible to generate an enormous number of different transformations while storing only relatively few transformations in the transformation function bank 205. Alternatively, each block can be generated from the first block of obscuring instructions using a single transformation function instead of the concatenated set of functions.

Page 13

The implementation of a compiler that preserves the ~~transportation~~ transformation information in this way will be known to those skilled in the art. By so preserving the transformation information, then transformation functions can be applied to the object level code to achieve compression, if desired.

Page 12

Obscuring code injector 301 injects the run time apparatus 302 comprising elements 303, 304, 305, 306 into the serialized critical function source code blocks 104 and the obscuring code blocks 206 to minimize the possibility for the serialized critical function source code to be observed. As a result, image 307 comprises multiple collections of blocks 302, 104, and 206~~7~~. At this stage, the pre-compilation obscured image 307 is ready to be compiled into object code. The obstruction compiler 308 is applied to pre-compilation obscured image 307 to create object level code blocks 309, 310, 311 in correspondence to blocks 302, 104 and 206. Each collections of a block 309, block 310, and 311 is referred to as an object level block O_i 312.

Page 15,

All executions of the code blocks are conducted within the memory address space indicated as 605. As the first step, O_{L1}701 is loaded at memory address L_1 711. tThis address will remain unchanged for all future O_{L2}, ..., O_{LN-1}, O_{LN} code blocks. CodeLoader 702 of O_{L1} executes within this space to allocate a dynamic memory location at address E_1, 710. It is important that address E_1, 710 beby dynamically assigned to ensure that the execution process of the code blocks cannot be automatically traced at a fixed address using conventional or commercially available tools. Because of the dynamic nature of this address allocation, address E_2 for the next set pf code blocks O_{L2} cannot be determined until the active instructions of O_{L1} descramble O_{L2}.

Page 16

Because the run time apparatus in this invention allows dynamic loading and execution of the blocks in data file, virtually any arbitrary number of obscuring instructions can be executed as long as execution overhead limit permits. Furthermore, because every block of instruction is executed at a dynamically assigned memory address, it makes tracing execution of these blocks a challenging task. Without highly specialized hardware devices, locating the address where a block of instructions is loaded in memory is virtually impossible. These characteristics of the runtime system ensure that obtaining and observing instructions in memory using tracing techniques are laborious and time consuming to the extent of being humanly impossible without the support of highly expensive and special designed hardware devices.